Architectural Risk Analysis

Paco Hope, Cigital, Inc. [vita[1]]
Steven Lavenhar, Cigital, Inc. [vita[2]]
Gunnar Peterson, Cigital, Inc. [vita[3]]

2005-10-03

L3 / L, M[4]

Architectural risk assessment is a risk management process that identifies flaws in a software architecture and determines risks to business information assets that result from those flaws. Through the process of architectural risk assessment, flaws are found that expose information assets to risk, risks are prioritized based on their impact to the business, mitigations for those risks are developed and implemented, and the software is reassessed to determine the efficacy of the mitigations.

Acknowledgements. Contributions and reviews by Niels J. Bjergstrom, Pamela Curtis, Robert J. Ellison, Dan Geer, Gary McGraw, C.C. Michael, John S. Quarterman, and Adam Shostack are gratefully acknowledged. Errors and omissions are the authors'.

Overview

This document begins with a definition of terms in the Software Risk Assessment Terminology[5] section. The Architectural Risk Management[6] section describes the actual process of risk management, which is broken down into the Asset Identification[7], Risk Analysis[8], and Risk Mitigation[9] sections. The emphasis is on risk analysis. The broader topic of risk management is specifically addressed in the Risk Management Framework[10] content area. Architectural Risk Assessment is a subset of the Risk Management Framework. This document gives some risk management context to show where the architectural risk assessment and analysis processes and artifacts fit in the larger risk management framework. This document specifically examines architectural risk analysis of software threats and vulnerabilities and assessing their impacts on assets.

Before discussing the process of software architectural risk assessment, it is helpful to establish the concepts and terms and how they relate to each other.

Software Risk Assessment Terminology

Risk assessment involves information assets, threats, vulnerabilities, risks, impacts, and mitigations. This section describes each of these concepts.

Information Assets

The process of risk management is centered around information assets. These are the resources that must be protected. Traditionally, security practitioners concern themselves with the confidentiality, integrity, availability, and auditability of information assets. Information assets vary in how critical they are to the business. Some organizations value confidentiality of data most highly, while others demand integrity and availability. In highly regulated contexts, it might be important to audit access and modification to sensitive

1. http://buildsecurityin.us-cert.gov/bsi/about_us/authors/546-BSI.html (Hope, Paco)
2. http://buildsecurityin.us-cert.gov/bsi/about_us/authors/197-BSI.html (Lavenhar, Steven)
3. http://buildsecurityin.us-cert.gov/bsi/about_us/authors/200-BSI.html (Peterson, Gunnar)
5. #dsy10-BSI_Terminology
6. #dsy10-BSI_ARM
7. #dsy10-BSI_AI
8. #dsy10-BSI_RA
9. #dsy10-BSI_RM
10. http://buildsecurityin.us-cert.gov/bsi/articles/best-practices/risk/250-BSI.html (Risk Management Framework (RMF))

information. Without knowing what assets need protection, and without knowing what happens when the protection fails, the rest of the risk analysis techniques cannot produce worthwhile results. An asset is referred to in threat analysis parlance as a threat target.

Threats

Threats are nouns: agents that violate the protection of information assets. Some threats are well known and obvious: crackers, disgruntled employees, criminals, and security auditing tools that probe potential vulnerabilities. Other threats are not conscious entities but must still be considered: hardware failures, performance delays, natural disasters, force majeure, and user errors. In many cases the software system does not have direct control of the threat and cannot prevent its actions but may only work to limit and contain the impact.

All categories of threats should be considered, but malicious and accidental human activities usually get the most attention. Note that not all threats exploit software failures. CERT and the U.S. Secret Service recently conducted a survey of companies that had experienced insider attacks. The survey concluded that "In 57% of the cases, the insiders exploited or attempted to exploit systemic vulnerabilities in applications, processes, and/or procedures (e.g., business rule checks, authorized overrides)" [1[11]]. Abusing an override mechanism that the user is authorized to use is not an abuse of the software—it is an abuse of trust placed in the person.

Vulnerabilities

Vulnerabilities take many forms, not just implementation bugs like the popular buffer overflow. Software can also be vulnerable because of a *flaw* in the architecture. Flaws are fundamental failures in the design that mean that the software always will have a problem no matter how well it is implemented. Ordinary *bugs*, on the other hand, are simply a failure to implement the architecture correctly. Reimplementing the broken code solves the problem. In the case of architectural flaws, however, significant redesign is usually necessary to solve the problem. Failure to encode quotation marks correctly could be a bug that makes a web application susceptible to SQL-injection attacks. A modification to the input filtering routine quickly eliminates the problem. Failure to authenticate between multiple cooperating applications, however, is an architectural flaw that cannot be trivially remedied. Unmitigated vulnerabilities require risk management planning to deal with impacts to assets.

Attacks

An attack occurs when an attacker acts and takes advantage of a vulnerability to threaten an asset.

Impacts

Impacts are consequences that the business must face if there is a successful attack. Some are expressed in terms of revenue: lost sales, corporate liability (e.g., Sarbanes-Oxley). Also important are impacts to the company's marketing abilities: brand reputation damage, loss of market share, failure to deliver services or products as promised. Secondary effects of software failures can include increased maintenance costs, increased customer support costs, longer time to market, legal, regulatory, and compliance impacts, and higher cost of development. Architectural risk analysis is performed to enable the business to manage its risk at a more granular level.

Risks

Risk is a product of the probability of a threat exploiting a vulnerability and the impact to the organization. The process of architecture risk management is the process of identifying those risks in software and then addressing them. Some complex risks spring to mind easily: a malicious attacker (threat) bypasses the authentication module (vulnerability) and downloads user accounts (information asset), thereby exposing the business to financial liability for the lost records (impact). It is important to note that the software

11. #dsy10-BSI_d0e108

architecture exists in a system context that includes risks in the physical, network, host, and data layers, and risks in those layers (including those generated outside the organization's perimeter) may cascade into the software architecture.

Mitigations

Risk mitigation refers to the process of prioritizing, implementing, and maintaining the appropriate risk-reducing measures recommended from the risk analysis process. Mitigating a risk means changing the architecture of the software or the business in one or more ways to reduce the likelihood or the impact of the risk. A mitigation consists of one or more *controls* whose purpose is to prevent a successful attack against the software architecture's confidentiality, integrity, and availability. For example, redundancy and diversity strategies may mitigate attacks against the system's availability.

Risk Analysis

Risk analysis is an activity geared towards assessing and analyzing system risks. Risk analysis can be conducted on a scheduled, event-driven, or as needed basis. Risk analysis can be implemented as an iterative process where information collected and analyzed during previous assessments are fed forward into future risk analysis efforts.

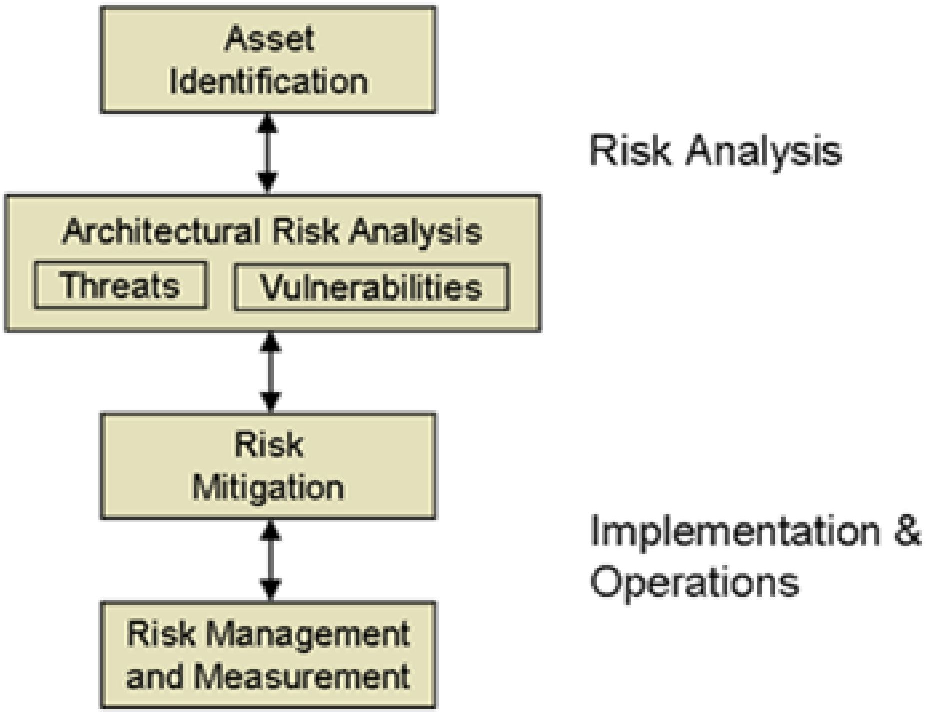
Risk Management

Risk management is an ongoing process that uses risk analysis, mitigations, metrics, and other processes and tools to manage risk for the organization. Risk management uses artifacts created in the risk analysis process to evaluate criteria that can be used to make risk management decisions. Risk management has an ongoing operational component where system and business metrics and events are monitored over time that may alter and evolve the organization's risk management posture to levels of risk that are acceptable to the organization. For example, Sarbanes-Oxley legislation altered the risk management reality for publicly traded organizations.

Architectural Risk Management

Risk management is the process of continually assessing and addressing risk throughout the life of the software. It encompasses four processes: (1) asset identification, (2) risk analysis, (3) risk mitigation, and (4) risk management and measurement. During each of these phases, business impact is the guiding factor for risk analysis. The architectural risk analysis process includes identification and evaluation of risks and risk impacts and recommendation of risk-reducing measures. The Risk Management Framework content area of this site contains more detail of the life cycle of risk management. This section focuses on risk management specifically related to software architecture.

The diagram below shows the process view of risk analysis and risk management areas.



Risk management is a continual process that regularly reevaluates the business's risks from software throughout the software's lifetime. The table below (taken from NIST SP800-34 [2[12]]) describes the risk management activities that take place at various times during the life cycle of a software system.

SLC Phase	Phase Characteristics	Risk Management Activities
Initiation	The need for software is expressed and the purpose and scope of the software is documented.	Information assets are identified. Business impacts related to violation of the information assets are identified. Risks are considered in the system requirements, including non-functional and security requirements, and a security concept of operations.
Development or Acquisition	The software is designed, purchased, programmed, developed, or otherwise constructed.	The risks identified during this phase can be used to support the security analyses of the software and may lead to architecture or design tradeoffs during development.
Implementation	The system security features are configured, enabled, tested, and verified.	The risk management process supports the assessment of the system implementation against its requirements and within its modeled operational environment. Decisions regarding risks identified must be made prior to system operation.

12. #dsy10-BSI_d0e175

Operation or Maintenance	The system performs its functions. Typically the system is being modified on an ongoing basis through the addition of hardware and software and by changes to organizational processes, policies, and procedures.	Risk management activities are performed for periodic system reauthorization (or reaccreditation) or whenever major changes are made to the software in its operational, production environment (e.g., new features or functionality).

Asset Identification

Risk management begins by identifying the assets that must be protected. Subsequent risk analysis depends on the accurate identification of the software's ultimate purpose and how that purpose ties into the business's activities. It is very often the case that software guards or uses information assets that are important to the business. These assets can be personal information about customers, financial information about the company itself, order information that the company needs in order to fulfill orders and collect revenue, or perhaps accounting information that must be managed carefully to comply with federal law.

To identify information assets, one must look beyond the software development team to the management that directs the software's evolution. That management determines what the software's goals are and what constraints it operates in. Furthermore, that management can identify the business impact of failures.

Information assets often take the form of databases, credentials (userid, password, etc.), audit records, financial information, intellectual property, and other vital business information. Each asset has different properties that are most important to it. For instance, integrity of audit records is most important (that none are added or deleted inappropriately, and that they are all accurate). In the case of financial records, confidentiality and integrity are very important, but if availability is negatively impacted, then business impact may manifest in other ways, such as lost customers or failure to meet Service Level Agreements.

Through a series of interviews with business representatives, the initial information regarding assets should be discovered. For each one, the business should identify the important properties to be maintained on that asset (e.g., confidentiality, auditability, integrity, availability) and the impact to the business if that property is not maintained.

Often assets can be identified through a thorough understanding of the software and how it does its work. For example, imagine that a customer service phone call increases in length by an average of 2 minutes when the phone routing software is unable to match the caller ID with the customer record. It is intuitively obvious that availability is important to the customer accounts database. It is further obvious that the company risks ill-will with its customers or must pay customer service representatives for extra time dealing with higher aggregate call volume when the software fails and remains unavailable for significant amounts of time.

Given the information assets, it should be relatively straightforward to consider what software modules manipulate those assets. Analysis should spiral outward from an asset to see what software reads, writes, modifies, or monitors that information. All the information assets that can be found should be gathered in a list to be coordinated with risk analysis.

Risk Analysis

Risk analysis is the second step in the risk management process. Potential threats are identified and mapped to the risk associated with them. The results of the risk analysis help identify appropriate controls for reducing or eliminating risk during the risk mitigation process. Risk is a function of the likelihood of a given threat exercising a particular potential vulnerability and the resulting impact of that adverse event on the organization or on information assets. In order to determine the likelihood of an adverse event occurring, threats to a system must be analyzed in conjunction with the potential vulnerabilities and the security controls in place for the system. Impact refers to the magnitude of impact that could be caused by a threat’s exercise of vulnerability. The level of impact is governed by the potential impacts to individuals or to the organization, its mission, or its assets and in turn produces a relative value for the IT assets and resources affected (e.g., the criticality and sensitivity of the software components and data). The risk assessment methodology encompasses six fundamental activity stages:

- application characterization
- architectural vulnerability assessment
- threat analysis
- risk likelihood determination
- risk impact determination
- risk mitigation

These activities are described below.

Application Characterization

Assessing the architectural risks for a software system is easier when the scope of the architecture is well defined. Defining its scope is the role of application characterization. The boundaries of the software system are identified, along with the resources, integration points, and information that constitute the system. Once the boundaries are defined, many artifacts are required or desired for review. These include, but are not limited to, the following:

• software business case • functional and non-functional requirements • enterprise architecture requirements • use case documents • misuse and abuse case documents • software architecture documents describing logical, physical, and process views • data architecture documents • detailed design documents such as UML diagrams that show behavioral and structural aspects of the system • software development plan • transactions	• security architecture documents • identity services and management architecture documents • quality assurance plan • test plan • risk management plan • software acceptance plan • problem resolution plan • risk list • issues list • project metrics • programming guidelines • configuration and change management plan • project management plan • disaster recovery plan • system logs • operational guides

It is often the case that a given software project does not have all of these artifacts. What is important is to collect as many as possible. Additional system-level artifacts are also useful in the architectural risk assessment process. These include

• documentation of the system and data criticality (e.g., the system's value or importance to the organization) • documentation of the system and data sensitivity • system security policies governing the software (organizational policies, federal requirements, laws, industry practices) • management controls used for the software (e.g., rules of behavior, security planning)	• information storage protection that safeguards system and data availability, integrity, and confidentiality • flow of information pertaining to the software (e.g., system interfaces, system input and output flowchart) • technical controls used for the software (e.g., built-in or add-on security products that support identification and authentication, discretionary

	or mandatory access control, audit, residual information protection, encryption methods)

For an application that is in the initiation or design phase, information necessary to perform the architectural risk assessment can be primarily derived from the design or requirements documents. For an application under development, it is necessary to define key security rules and attributes. System design documents and the system security plan can provide useful information about the security of software in the development phase. For software that has been fielded, data is collected about the software in its production environment, including data on system configuration, connectivity, and documented and undocumented procedures and practices. The system description is informed by the underlying security infrastructure or future security plans for the software.

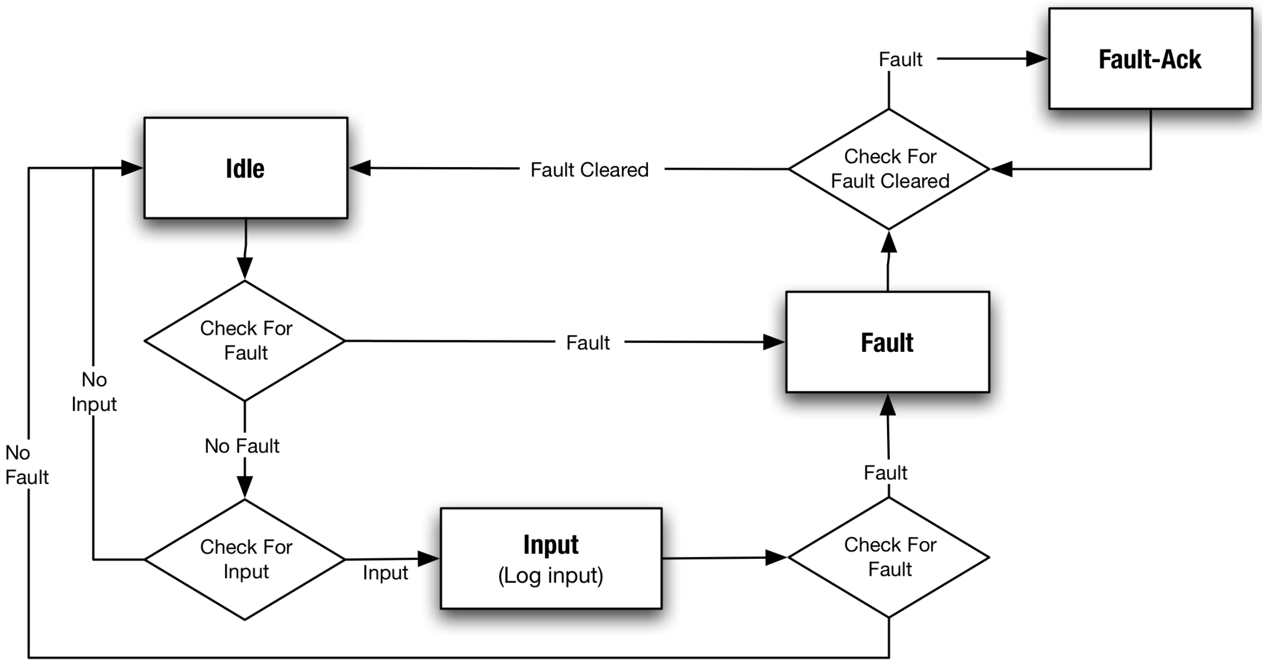
In addition to reviewing the SDLC artifacts, questionnaires and interviews are useful in gathering information relevant to the risk assessment of the application. Policy documents, system documentation, and security-related documentation such as audit reports, risk assessment reports, system test results, system security plans, and security policies can also provide important information about the security controls used by and planned for the software.

In cases where the application is already in production or uses resources that are in production such as databases, servers, identity systems, and so on, these systems may have already been audited and assessed. These assessments, when they exist, may provide a rich set of analysis information.

In the end, the goal of the application characterization activity is to produce one or more documents that depict the vital relationships between critical parts of the system. It is often not practically possible to model and depict all interrelationships. Using information gathered through asset identification and from security best practices, the diagrams and documents gradually take shape.

Sometimes processes are depicted using a state diagram, in order to validate that all states are covered by code, by tests, or by requirements. Figure 1[13], for example, depicts a software process that constantly checks for faults or inputs and then waits for faults to be cleared by manual intervention. Such a diagram would be a small part of a much larger overall system architecture and would only be diagrammed to this level of detail if it were protecting an important information asset that was the subject of some scrutiny.

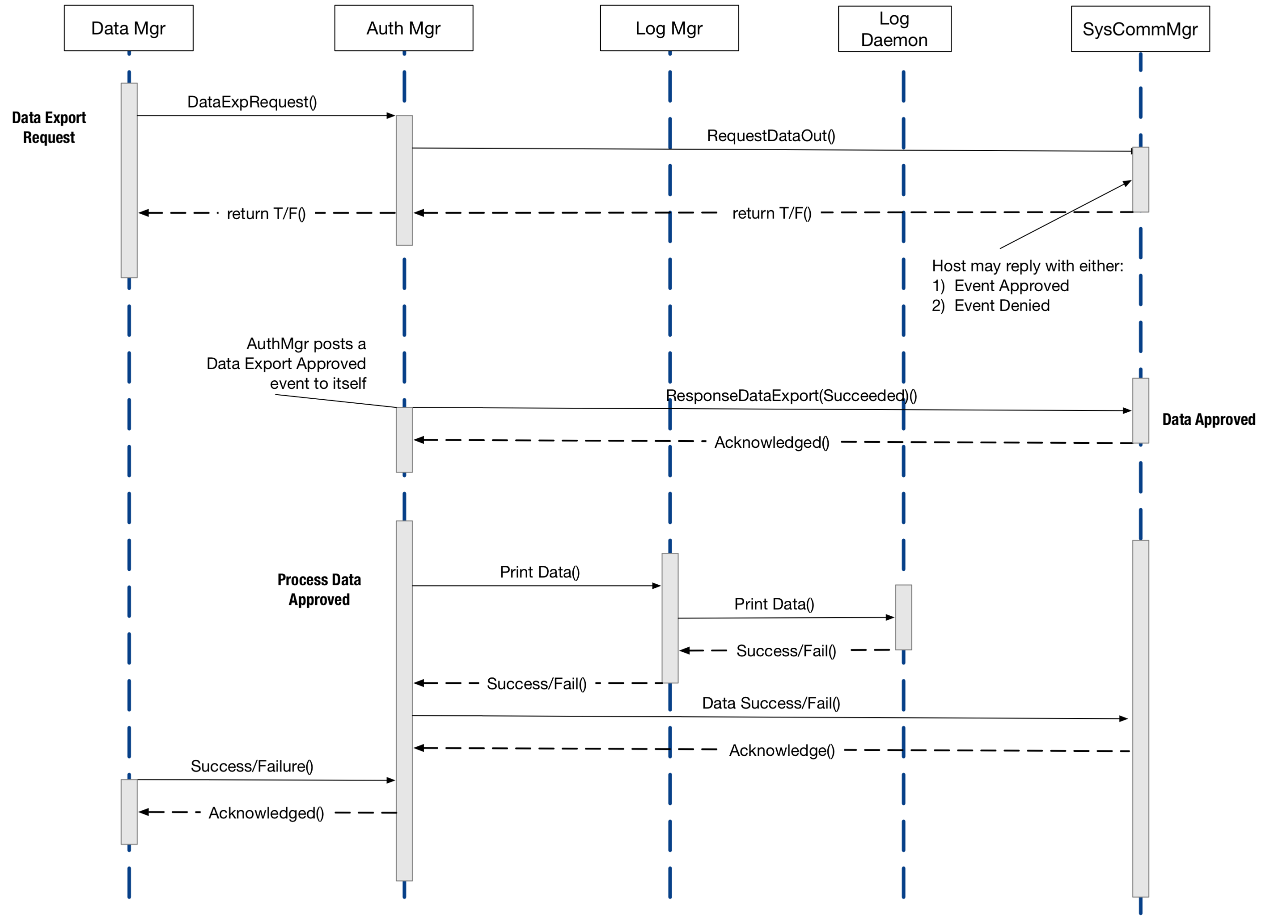
Figure 1. State diagram



13. #dsy10-BSI_Figure-1

At other times, complex communication needs to be depicted using an interaction diagram to determine potential opportunities for attack. Figure 2[14] shows a set of five processes that intercommunicate to determine whether data may be exported. In the event that data is exported, a logging subsystem is activated to write log entries to record the fact that data was exported.

Figure 2. Data export message passing between five processes



Architectural Risk Analysis

Three activities can guide architectural risk analysis: known vulnerability analysis, ambiguity analysis, and underlying platform vulnerability analysis. Architectural risk analysis examines the preconditions that must be present for vulnerabilities to be exploited and assesses the states that the system may enter upon exploitation. As with any quality assurance process, risk analysis testing can only prove the presence, not the absence, of flaws. Architectural risk analysis studies vulnerabilities and threats that may be malicious or non-malicious in nature. Whether the vulnerabilities are exploited intentionally (malicious) or unintentionally (non-malicious) the net result is that the confidentiality, integrity, and/or availability of the organization's assets may be impacted. Risk management and risk transfer instruments deal with unmitigated vulnerabilities.

One of the strengths of conducting risk analysis at the architectural level is to see the relationships and impacts at a system level. In practice, this means assessing vulnerabilities not just at a component or function level, but also at interaction points. Use case models help to illustrate the relationships among system components. The architecture risk analysis should factor these relationships into the vulnerabilities analysis and consider vulnerabilities that may emerge from these combinations.

14. #dsy10-BSI_Figure-2

Known Vulnerability Analysis

There are a lot of known vulnerabilities documented throughout software security literature. They range from the obvious (failure to authenticate) to the subtle (symmetric key management). Static code checkers, runtime code checkers, profiling tools, penetration testing tools, stress test tools, and application scanning tools can find some security bugs in code, but they do not address architectural problems. For example, a static code checker can flag bugs like buffer overflows. It cannot identify security vulnerabilities like transitive trust.

When performing known vulnerability analysis, consider the architecture as it has been described in the artifacts that were reviewed for asset identification. Consider it against a body of known bad practices or known good principles for confidentiality, integrity, and availability. For example, the good principle of "least privilege" prescribes that all software operations should be performed with the least possible privilege required to meet the need. To consider architecture in light of this principle, find all the areas in the system that operate at an elevated privilege. Perhaps diagram the system's major modules, classes, or subsystems and circle areas of high privilege versus areas of low privilege. Consider the boundaries between these areas and the kinds of communications across those boundaries.

Ambiguity Analysis

Ambiguity is a rich source of vulnerabilities when it exists between requirements or specifications and development. Architecture's role is to eliminate the potential misunderstandings between business requirements for software and the developers' implementation of the software's actions. It is vital to acquire business statements (marketing literature, business goal statements, etc.) and requirements-phase artifacts (use cases, user stories, requirements). These pre-requirement and requirement artifacts must be contrasted with development artifacts (code, low-level design, API documentation) and then compared to the intermediate architecture documentation.

The important point is to note places where the requirements are ambiguously stated and the implementation and architecture either disagree or fail to resolve the ambiguity. For example, a requirement for a web application might state that an administrator can lock an account and the user can no longer log in while the account remains locked. What about sessions for that user that are actively in use at the time the administrator locks the account? Is the user suddenly and forcibly logged out, or is the active session still valid until the user logs out? The authentication and authorization architecture must be compared to the actual implementation to learn which way this question was decided. The security ramifications of logins that persist even after the account is locked should be considered against the sensitivity of the information assets being guarded.

Underlying Platform Vulnerability Analysis

An architectural risk assessment must include an analysis of the vulnerabilities associated with the application's execution environment. This will include operating system vulnerabilities, network vulnerabilities, platform vulnerabilities (popular platforms include WebLogic, WebSphere, PHP, ASP.net, and Jakarta), and interaction vulnerabilities resulting from the interaction of components. The goal of this step is to develop a list of application or system vulnerabilities that could be accidentally triggered or intentionally exploited and result in a security breach or a violation of the system’s security policy. When credible threats can be combined with the vulnerabilities uncovered in this exercise, a risk exists that needs further analysis and mitigation.

The types of vulnerabilities that will exist and the methodology needed to determine whether the vulnerabilities are present will vary depending on which phase in the SDLC the risk assessment occurs. In the requirements phase, the search for vulnerabilities should focus on the organization’s security policies, planned security procedures, non-functional requirement definitions, use cases, and misuse and abuse cases. In the implementation phase, the identification of vulnerabilities should include more specific information, such as the planned security features described in the security design documentation. For fielded applications that are operational, the process of identifying vulnerabilities should include an analysis of the software security features and the security controls, technical and procedural, used to protect the system. Fielded systems can also use the results of system tests and reports from users in the field to identify problems.

Independent of the life-cycle phase, online vulnerability references should be consulted. Every application platform and operating system has a mailing list and a web site where up-to-date vulnerability information can be found. There are also several web sites that aggregate vulnerability information. These sites and lists should be consulted regularly to keep the vulnerability list current for a given architecture.

Vulnerability Classification

Classifying vulnerabilities allows for pattern recognition of vulnerability types. This in turn may enable the software development team to recognize and develop countermeasures to deal with classes of vulnerabilities by dealing with the vulnerabilities at a higher level of abstraction. The RISOS Study [3[15]] detailed seven vulnerability classes:

- incomplete parameter validation: input parameters not validated for type, format, and acceptable values
- inconsistent parameter validation: input validation does not follow consistent scheme
- implicit sharing of privileged/confidential data: resources are not appropriately segregated
- asynchronous validation/inadequate serialization: vulnerabilities resulting from concurrency, sequencing of events as in message queue systems
- inadequate identification/authentication/authorization: access control vulnerabilities
- violable prohibition/limit: lack of enforcement on resource limitations, such as buffer overflows
- exploitable logic error: program logic errors enabling circumvention of access control

Threat Analysis

Threats are agents that violate the protection of information assets and site security policy. Threat analysis identifies for a specific architecture, functionality and configuration. Threat analysis may assume a given level of access and skill level that the attacker may possess. Threats may be mapped to vulnerabilities to understand how the system may be exploited. A mitigation plan is composed of countermeasures that are considered to be effective against the identified vulnerabilities that the threats exploit.

Attackers who are not technologically sophisticated are increasingly performing attacks on systems without really understanding what it is they are exploiting, because the weakness was discovered by someone else. These individuals are not looking to target specific information or a specific company but rather use knowledge of a vulnerability to scan the entire Internet[16] for systems that possess that vulnerability. The table below, which was developed by NIST [4[17], p. 14], summarizes potential threat sources:

Threat Source	**Motivation**	**Threat Actions**
Cracker	Challenge Ego Rebellion	• System profiling • Social engineering • System intrusion, break-ins • Unauthorized system access
Computer criminal	Destruction of information Illegal information disclosure Monetary gain Unauthorized data alteration	• Computer crime (e.g., cyber stalking) • Fraudulent act (e.g., replay, impersonation, interception) • Information bribery • Spoofing • System intrusion • Botnets

15. #dsy10-BSI_d0e468
16. http://www.webopedia.com/TERM/S/script_kiddie.html##
17. #dsy10-BSI_nist

		• Malware: Trojan, virus, worm, spyware • Spam • Phishing
Terrorist	Blackmail Destruction Exploitation Revenge Monetary gain Political gains	• Bomb • Information warfare • System attack (e.g., distributed denial of service) • System penetration • System tampering
Industrial espionage	Competitive advantage Economic espionage Blackmail	• Economic exploitation • Information theft • Intrusion on personal privacy • Social engineering • System penetration • Unauthorized system access (access to classified, proprietary, and/or technology-related information)
Insiders (poorly trained, disgruntled, malicious, negligent, dishonest, or terminated employees)	Curiosity Ego Intelligence Monetary gain Revenge Unintentional errors and omissions (e.g., data entry errors, programming errors) Wanting to help the company (victims of social engineering) Lack of procedures or training	• Assault on an employee • Blackmail • Browsing of proprietary information • Computer abuse • Fraud and theft • Information bribery • Input of falsified, corrupted data • Interception • Malicious code (e.g., virus, logic bomb, Trojan horse) • Sale of personal information • System bugs • System intrusion • System sabotage • Unauthorized system access

An issue that greatly complicates the prevention of threat actions is that the basic intent of the attack often cannot be determined. Both internal and external threat sources may exist, and an attack taxonomy should differentiate between attacks that require insider access to a system and attacks initiated by external sources. Internal attacks may be executed by threat actors such as disgruntled employees and contractors. It is important to note that nonmalicious use by threat actors may result in system vulnerabilities being exploited. Internal threat actors can act on their own or under the direction of an external threat source (for example, an employee may install a screensaver that contains a Trojan horse). Internal threat agents currently account for the majority of intentional attacks against government and commercial enterprises.

Some threat actors are external, and may include structured external, transnational external, and unstructured external threats, which are described below.

- Structured external threats are generated by a state-sponsored entity, such as a foreign intelligence service. The resources supporting the structured external threat are usually quite high and sophisticated.
- Transnational threats are generated by organized non-state entities, such as drug cartels, crime syndicates, and terrorist organizations. Such threats generally do not have as many resources as the structured threats (although some of the larger transnational threat organizations may have more resources than some smaller, structured threat organizations). The nature of the transnational external threat makes it more difficult to trace and provide a response. Transnational external threats can target members or staff of the Treasury employing any or all of the techniques mentioned above.
- Unstructured external threats are usually generated by individuals such as crackers. Threats from this source typically lack the resources of either structured or transnational external threats, but nonetheless may be very sophisticated. The motivation of such attackers is generally, but not always, less hostile than that underlying the other two classes of external threat. Unstructured threat sources generally limit their attacks to information system targets and employ computer attack techniques. New forms of loosely organized virtual hacker organizations ("hacktivists - hackers and activists") are emerging.

Mapping Threats and Vulnerabilities

The combination of threats and vulnerabilities illustrates the risks that the system is exposed to. Shirey [5[18]] provides a model of risks to a computer system related to disclosure, deception, disruption, and usurpation. Threats may target these risk classes:

- Disclosure: the dissemination of information to an individual(s) for whom the information should not be seen.
- Deception: risks that involve unauthorized change and reception of malicious information stored on a computer system or data exchanged between computer systems.
- Disruption: where access to a computer system is intentionally blocked as a result of an attack or other malicious action. It is important to note that in some cases performance degradation can be as harmful as performance interruption.
- Usurpation: unauthorized access to system control functions.

Risk classification assists in communication and documentation of risk management decisions. Threats and vulnerabilities conspire to participate in one or more risk categories. Risk mitigation mechanisms deal with one or more risk categories. Threats and vulnerabilities may combine to create additional weaknesses in the system.

Risk Likelihood Determination

Having determined what threats are important and what vulnerabilities might exist to be exploited, it can be useful to estimate the likelihood of the various possible risks. In software security, "likelihood" is a qualitative estimate of how likely a successful attack will be, based on analysis and past experience. Since it is based on past experience, this likelihood cannot account for new types of attacks or vulnerabilities that have not yet been discovered. It might not accurately reflect the *probability* of a successful attack. Nonetheless, the concept of likelihood can be useful when prioritizing risks and evaluating the effectiveness of potential mitigations.

The following factors must be considered in the likelihood estimation:

- the threat's motivation and capability
- the vulnerability's directness and impact
- the effectiveness of current controls

The threat's motivation and capability vary widely. A college student who hacks for the fun of it is less motivated than a paid hacker who has backing or the promise of a significant payment. A former employee

18. #dsy10-BSI_d0e749

who has a specific grievance against a company will be more motivated and informed than an outsider who has no special knowledge of the target system's internal workings.

Some vulnerabilities are direct and have severe impacts. For example, a vulnerability is very direct and severe if it allows a database server to be compromised directly from the Internet using a widely distributed exploit kit. An indirect vulnerability that is less severe is one that requires an exploit payload to pass unmodified through several different systems only to produce a log entry that might cause an unexpected failure in the logging system.

The effectiveness of current controls characterizes how high the bar is set for an intentional attacker or how unlikely an accidental failure is. For example, simple userids and passwords can be compromised much more easily than most two-factor authentication systems. Adding a second authentication factor raises the bar for a would-be threat. However, if the second factor in the authentication is a biometric thumbprint reader that can be spoofed with latent image recovery techniques, the additional controls are not as effective.

The likelihood is a subjective combination of these three qualities (motivation, directness of vulnerability, and compensating controls). These can be boiled down to a rating of high, medium, or low. The likelihood levels are described in the table below.

High	The three qualities are all weak: a threat is highly motivated and sufficiently capable, a vulnerability exists that is severe and direct, and controls to prevent the vulnerability from being exploited are ineffective.
Medium	One of the three qualities is compensating, but the others are not. The threat is perhaps not very motivated or not sufficiently capable, the controls in place may be reasonably strong, or the vulnerability might be indirect or not very severe.
Low	Two or more of the three qualities are compensating. The threat might lack motivation or capability. Strong controls might be in place to prevent, or at least significantly impede, the vulnerability from being exploited. The vulnerability might be very indirect or very low impact.

Risk Impact Determination

Independent of likelihood and controls, the risk's impact must be determined. That is, what consequences will the business face if the worst-case scenario in the risk description comes to pass. Furthermore, the analysis must account for other credible scenarios that are not the worst case yet are bad enough to warrant attention. Below we discuss three aspects of risk impact determination: identifying the threatened assets, identifying business impact, and determining impact locality.

Identify Threatened Assets

The assets threatened by the impact of this risk, and the nature of what will happen to them, must be identified. Common impacts to information assets include loss of data, corruption of data, unauthorized or unaudited modification of data, unavailability of data, corruption of audit trails, and insertion of invalid data.

Identify Business Impact

The business will suffer some impact if an attack takes place. It is of paramount importance to characterize that impact in as specific terms as possible. Risk management efforts are almost always funded ultimately by management in the organization whose primary concern is monetary. Their support and understanding can be assured only by driving software risks out to fiscal impacts. If the worst possible consequence of a software failure is the loss of $10,000 to the business, but it will take $20,000 in labor hours and testing to fix the software, the return on investment for mitigation does not make financial sense. Furthermore, correct financial assessment of impact drives prioritization. It is usually more important to fix a flaw that can precipitate a $25 million drop in the company's market capitalization before fixing a flaw that can expose the business to a regulatory penalty of $500,000. Unless software risks are tied to business impacts, however, such reasoning is not possible.

Example business impacts include failing to control access to medical records, thus exposing the business to liability to lawsuits under the Health Insurance Portability and Accountability Act (HIPAA); and a race condition in order insertion and order fulfillment operations on the orders database that causes orders to be duplicated or lost.

Impact Locality

All impacts will have a locality in space, time, policy, and law. In addition to characterizing the monetary impact, the location in other dimensions may be useful or required. For example, if an encryption key is stored unencrypted, it matters whether that key is in the dynamically allocated RAM of an application on a trusted server, or on the hard disk of a server on the Internet, or in the memory of a client application. Likewise, laws and policies apply differently depending on where data is stored and how data exposures happen.

Impacts can sometimes be localized in time or within business and technical boundaries. For example, a failure in the application server might only prevent new orders from being placed, while orders that are already placed can be fulfilled and customer service staff can see, modify, and update existing orders. Such an impact is localized in time and in a fraction of the merchandising side of the business.

There are two special types of impact classes to consider that may have a more global impact. One is risks that may impact a domain system, such as a national or enterprise-wide system, that is by its nature a single point of failure (for example, a Red Telephone that fails to ring). The other concerns cascade failure, where failures in a technical system like the Domain Name Service or a business system like the general ledger may cascade across other systems and domains.

Technical risk impact determination is supported by the artifact analysis. There are a number of processes available for software risk identification, including the use of automated tools and the application of checklists and guidelines. The method used should strive to quantify risks in concrete terms. Examples of artifact quality metrics include, but are not limited to, number of defects, number of critical risks, identified risks by type, and progress against acceptance criteria.

As with risk likelihood, subjective High, Medium, and Low rankings may be used to determine relative levels of risk for the organization.

Risk Exposure Statement

The risk exposure statement combines the likelihood of the risk occurring with impact of the risk. The product of these two sets of analysis provides the overall summary of risk exposure for the organization for each risk. The table below describes a method of generating the risk exposure statement.

<table>
<tr><td rowspan="3">Likelihood</td><td>High</td><td>Medium</td><td>High</td><td>High</td></tr>
<tr><td>Medium</td><td>Low</td><td>Medium</td><td>High</td></tr>
<tr><td>Low</td><td>Low</td><td>Low</td><td>Medium</td></tr>
<tr><td rowspan="2" colspan="2"></td><td>Low</td><td>Medium</td><td>High</td></tr>
<tr><td colspan="3">Impact</td></tr>
</table>

The risk exposure statement generalizes the overall exposure of the organization for the given risk and offers more granular visibility to both impact and likelihood. The risk exposure statement gives the organization more fine grained control over risk management but does not require all risks to be eliminated. Alan Greenspan, Chairman of the Federal Reserve Board, said this in 1994:

There are some who would argue that the role of the bank supervisor is to minimize or even eliminate bank failure; but this view is mistaken in my judgment. The willingness to take risk is essential to the growth of the *free market economy…[i]f all savers and their financial intermediaries invested in only risk-free assets, the potential for business growth would never be realized* [6[19]].

Risk Mitigation

The various risks that have been identified and characterized through the process of risk analysis must be considered for mitigation. Mitigation of a risk means to change the architecture of the software or the business in one or more ways to reduce the likelihood or the impact of the risk. Formal and informal testing, such as penetration testing, may be used to test the effectiveness of the mitigations. Although changing how the business operates (e.g., insuring against impacts of risks) is a valid response to risk, it is outside the scope of architecture assessment, so it will not be covered here.

Mitigations to architectural flaws are almost always much more complicated than mitigating implementation bugs. They often require cooperation between multiple modules, multiple systems, or at least multiple classes; and the cooperating entities may be managed and implemented by different teams. Thus, when a flaw is found, the fix usually requires agreement across multiple teams, testing of multiple integrated modules, and synchronization of release cycles that may not always be present in the different modules.

Reducing the likelihood of a risk can take several forms. "Raising the bar" in terms of the skills necessary to exploit a vulnerability is often a first step. For example, changing authentication mechanisms from userid and password to pre-shared public key certificates can make it far more difficult to impersonate a user. Reducing the period of time that a vulnerability is available for exploit is another way to reduce the likelihood of a risk. If sessions expire after 10 minutes of inactivity, then the window of opportunity for session hijacking is about 10 minutes long.

Reducing the impact of a risk can also take several forms. Most developers immediately consider eliminating the vulnerability altogether or fixing the flaw so that the architecture cannot be exploited. Cryptography can help, for example, when applied correctly. It is easier to detect corruption in encrypted data than in unencrypted data, and encrypted data is harder for an attacker to use if they get it. Sometimes, from a business point of view, it makes more sense to build functionality that logs and audits any successful exploits. Remediating a broken system might be too expensive, whereas adding enough functionality to have a high probability of stopping an exploit in progress might be sufficient.

Many mitigations can be described either as detection or correction strategies. Depending on the cost of making failure impossible through correction, it may be much more cost effective to enable systems to detect and repair failure quickly and accurately. Imagine a software module that is very temperamental and tends to crash when provided bad input and (for the sake of argument) cannot be modified or replaced. A focus on correction would add business logic to validate input and make sure that the software module never received input that it could not handle. In contrast, a focus on correction would add monitoring or other software to watch for the module to crash and try to restart the module quickly with minimal impact.

Mitigation is never without cost. The fact that remediating a problem costs money makes the risk impact determination step even more important to do well. Mitigations can often be characterized well in terms of their cost to the business: man-hours of labor, cost of shipping new units with the improved software, delay entering the market with new features because old ones must be fixed, etc. This ability to characterize the mitigation's cost, however, is of little value unless the cost of the business impact is known.

It is important to note that risk mitigation mechanisms may introduce threats and vulnerabilities to the system, and as such need to be analyzed. The risk analysis process is iterated to reflect the mitigation’s risk profile.

Risk Management and Measurement

Due to cost, complexity, and other constraints, not all risks may be mitigated. Organizations may seek to accept the risk as a “cost of doing business,” or they may choose to outsource risk via insurance or contractual means, or the risk may be mitigated partially. Risk management categorizes the controls that

19. #dsy10-BSI_d0e935

mitigate risks and tracks their efficacy over time through testing, log analysis, auditing, and other means. Risk measurement is a tool used to monitor the risk exposure to the organization over time. Metrics provide quantitative analysis information that may be used to judge the relative resilience of the system over time. Andrew Jaquith [7[20]] provides guidelines that security metrics must adhere to:

- **Be consistently measured.** The criteria must be objective and repeatable.
- **Be cheap to gather.** Using automated tools (such as scanning software or password crackers) helps.
- **Contain units of measure.** Time, dollars, or some numerical scale should be included—not just, say, "green," "yellow" or "red" risks.
- **Be expressed as a number.** Give the results as a percentage, ratio, or some other kind of actual measurement. Don't give subjective opinions such as "low risk" or "high priority."

Ongoing objective measurement provides insight into the effectiveness of the risk management decisions and enables improvement over time. While the software industry as a whole currently lacks agreed-upon standards for precise interval scale metrics, software teams can adopt ordinal scale metrics that place events, controls, and security posture on a continuum. Ordinal scale metrics provide data that can be used to drive decision support by allowing visibility and modeling of the ranking of security metrics. Security metrics collection and analysis benefits from consistency; although the measurements may emphasize certain aspects of the problem (counting lines of code to gauge complexity) while ignoring other aspects of the problem (interfaces to code), the trend data gained by using consistent measures remains valuable.

Summary

Risk management is composed of point-in-time and ongoing processes. As the software evolves, its architecture must be kept up to date. The body of known attack patterns is always growing, thus continued success in known vulnerability analysis is dependent on remaining current in software security trends. Ambiguity analysis is always necessary, though over time it can focus on just new requirements or new functionality that is being added. Even with that focus, it is worthwhile to occasionally step back and reappraise the entire system for ambiguity. As platforms upgrade and evolve, each subsequent release will fix older problems and probably introduce new ones. Thus underlying platform vulnerability analysis must continue throughout the life of the product.

A master list of risks should be maintained during all stages of the architectural risk analysis. It should be continually revisited to determine mitigation progress and help improve processes on future projects. For example, the number of risks identified in various software artifacts and/or software life-cycle phases is used to identify problematic areas in software process. Likewise, the number of risks mitigated over time is used to show concrete progress as risk mitigation activities unfold. Ideally, the display and reporting of risk information should be aggregated in some automated way and displayed in a risk dashboard that enables accurate and informed decisions.

Future Directions

As risk management continues to evolve to keep pace with technology and business realities, two websites that track emerging issues closely are Security Metrics (http://www.securitymetrics.org) a website and wiki devoted to security analysis driven by metrics, and Perilocity (http://riskman.typepad.com/perilocity/), which is a blog focused on Internet risk management.

References

[1] Michelle Keeney, JD, PhD, et al. *Insider Threat Study: Computer System Sabotage in Critical Infrastructure Sectors*, May 2005, http://www.secretservice.gov/ntac_its.shtml.

[2] M. Swanson, A. Wohl, L. Pope, T. Grance, J. Hash, R. Thomas, “Contingency Planning Guide for Information Technology Systems,” NIST (2001).

20. #dsy10-BSI_d0e963

[3] R. Abbott, J.Chin, J. Donnelley, W. Konigsford, S. Tokubo, and D. Webb, “Security Analysis and Enhancements of Computer Operating Systems,” Technical Report NBSIR 76-1041, ICET, National Bureau of Standards, Washington, DC 20234 (Apr. 1976).

[4] National Institute of Standards and Technology. Risk Management Guide for Information Technology Systems (NIST 800-30). http://csrc.nist.gov/publications/nistpubs/800-30/sp800-30.pdf (2002).

[5] R. Shirey, *Security Architecture for Internet Protocols: A Guide for Protocol Designs and Standards,* Internet Draft: draft-irtf-psrg-secarch-sect1-00.txt (Nov. 1994).

[6] Address to the Garn Institute of Finance, University of Utah, November 30, 1994.

[7] Andrew Jaquith, Yankee Group, CIO Asia, “A Few Good Metrics”, http://cio-asia.com/ShowPage.aspx?pagetype=2&articleid=2560&pubid=5&issueid=63 (2005).

Cigital, Inc. Copyright

The Build Security In (BSI) portal is sponsored by the U.S. Department of Homeland Security (DHS), National Cyber Security Division. The Software Engineering Institute (SEI) develops and operates BSI. DHS funding supports the publishing of all site content.

1. mailto:copyright@cigital.com